Final Report

by
Yun Zhou

CS 297
Spring 2004

Advisor: Dr. Pollett

Table of Contents

I. Introduction

Mozilla is probably one of the biggest open source projects at the time of this writing, and its codebase keeps growing as more functionalities are integrated. Mozilla implements a full-fledged Internet browser with strong security protection, high performance, and good usability.

1.1 Strengths and Weaknesses of Mozilla's Profile Manager

Profile manager is one of the key components of Mozilla. Mozilla uses profiles to store users' personal settings such as bookmarks, cookies, preferences, news group subscriptions, browsing history, cache, and so forth. Profiles not only help users manage their personal information, customize the browser's behavior in response to different events, but also provide big performance gain. For example, the usage of cookies and cache largely reduce the number of message exchanges between the browser and the content servers. Users are allowed to create multiple profiles and save them to user-specified locations, such as removable disks. The profile manage provides a user interface for the users to manage their profiles. The content of the a profile can be easily accessed through the browser's link field. In a following section, I will explain in more detail how to examine a profile.

However, the current profile manager has the following weaknesses that need improving. First, although profiles can be stored on removable storage medium, they do not really achieve location independence. This is because Mozilla does not recognize profiles unless they are registered. Registration can only be done when a profile is created, meaning that one cannot easily take a profile created on one computer and use it on another without copying and pasting all the files in the profile folder.

Second, the files in a profile folder is saved without any encryption and the profile folder is given a software-generated folder name ended with ".slt". Therefore, if an unauthorized user does a simple file search for folders with the pattern ".slt", she can easily locate the profiles and examine the content. This can be a security flaw because

not only can private information be leaked, but also a hacker or malicious code and change certain security settings such as disabling strong encryption ciphers, allowing no encryption at all, or disabling pop-up warnings when a suspicious events occurs.

1.2 Purpose of the Project for CS 297 and CS 298

The USB (Universal Serial Bus) key profile manager project is meant to improve the current profile manager of Mozilla by overcoming the weaknesses listed in the previous section without significant influence to the performance of the browser.

To achieve location independence, the USB profile manager can automatically detect a mounted USB drive and search for profiles according to certain pattern. If profiles are detected, they will be registered with the browser so that the user can use them as regular profiles. When the browser is shut down, the loaded profiles will be removed from the registry so that no footprint will be left. The process should be transparent to the user without explicit copying and pasting.

To protect the content of the profiles, Advanced Encryption Scheme (AES) is considered to be used for encryption. User authentication can be enforced by a prompt for password. This feature will be implemented in the next version.

This project will be developed on Red Hat Linux 9.0. The current version of Mozilla is 1.7 RC1. The USB key profile manager will be implemented as an XPCOM component for backward compatibility. No modification will be made to the existing code base of Mozilla. After installation, the new component will be automatically registered with the browser and start functioning. It can easily uninstalled as well.

1.3 Outline of This Report

In this report, I will present the performance test of the USB drive I used for this project. Then I will discuss Mozilla's profile manager and Personal Security Manager (PSM). The implementation of the profile loading feature of the USB Profile Manager will be covered in section V. Finally, I will briefly describe the Advanced Encryption

Standard (AES).

II. Performance Test of a USB Drive

2.1 Advantages and Disadvantages of USB Drives

USB drives have gained more and more popularity due to their larger storage space, higher data transfer rate, smaller physical size, and prettier outlook compared to traditional floppy disks. With the advent of the 2.0 USB interface, the transfer rate will be much higher. In addition to these advantages, USB drives are also favored due the fact that newly manufactured computers usually don't come with a floppy drive any more.

2.2 Several USB Drives, Their Storage Space and Transfer Rates

The two characteristics of USB drives that I am mainly concerned about in this project are storage space and transfer rate. Here is a list of USB drive specifications.

Table1: Several USB drive products, storage space, and transfer rates

Brand	***Storage Space***	***Transfer Rate***
Kingston Traveler USB 2.0 Hi-Speed Flash Memory	1GB	480 mbps
Lexar Media USB JumpDrive 2.0 Pro	1GB	4.5/6.0 mbps
Sony Corporation Micro Vault USB Storage Media	256 MB	5.5 mbps
*PNY Attache 2.0 USB Flash Drive	128 MB	4.0/5.0 mbps
CD Cyclone Flash Key	128 MB	800 kbps
EasyDisk USB Drive	128 MB	710 kbps
Sonnet Technologies Piccolo USB Flash Drive	256 MB	350/700 kbps
Linksys Instant USB Disk	128 MB	800 kbps

Note: * indicates the USB Drive being used for this projects.

Source: http://accessories.us.dell.com/sna/productlisting.aspx?c=us&l=en&cs=RC968571&category_id=5949&first=true

The USB drives with lower transfer rates are most likely 1.1 USB drives. However, for 2.0 USB drives, if the underlying operating system does not support 2.0 USB

interface, the transfer rate will be that of a 1.1 USB interface.

2.3 Read and Write Performance Test

The following table shows the different tests I have carried out with regard to the read and write performance of a PNY Attache 128 MB 2.0 USB Flash Drive. The rates listed were the average rates of five experiments.

Table 2: Read/Write performance test results

	Write Rate (Average)	***Read Rate (Average)***
USB Drive	7.168 mbps	4.944 mbps
Local Drive	83.152 mbps	156.32 mbps
Local Drive/USB Drive	11 times faster	28.8 times faster

Although USB drives are much faster than floppy drives, they are still significantly lower in performance than local hard disks. However, as we can see from Table 1, faster 2.0 USB drives are coming out to the market. Although they are still costly at the moment, they will certainly be affordable in the near future.

III. Mozilla's Cache and Profile Manager Implementation

3.1 Examining the Cache

As of the current version of Mozilla, both sensitive and non-sensitive data can be cached. There are two types of caches: memory cache and disk cache. The memory cache is mainly for data marked explicitly not to be stored on disk. The disk cache is the cache stored in the folder of the profile in use. In addition to these two caches, the browse also uses the operating system's temporary folder for storing data until the user specifies a destination folder.

Mozilla provides an easy way for the user to look into the cache content. She just needs to type “about:cache” in the link field. The following is a sample listing of the cache usage:

Memory cache device

Number of entries:	126
Maximum storage size:	4194304 Bytes
Storage in use:	3375230 Bytes

Memory cache usage report:
List Cache Entries

Disk cache device

Number of entries:	692
Maximum storage size:	51200000 Bytes
Storage in use:	5784064 Bytes
Cache Directory:	/root/.mozilla/default/rqzr6l8u.slt/Cache

List Cache Entries

The the link "List Cache Entries" will bring the user more detailed information about the cache. For example,

Key: http://www.pny.com/images/white.gif
Data size: 43 Bytes
Fetch count: 1
Last Modified: Sat 15 May 2004 09:33:10 PM PDT
Expires: Sun 16 May 2004 09:33:09 PM PDT

Key: http://www.google.com/nav_page.gif
Data size: 373 Bytes
Fetch count: 1
Last Modified: Sat 15 May 2004 06:12:08 PM PDT
Expires: Sun 17 Jan 2038 11:14:06 AM PST

3.2 Implementation of the Profile Manager

As I mentioned earlier, profiles must be registered first in order to be useful. Mozilla has a special registry file for profiles. The major functionalities of the profile manager are implemented in nsProfile.cpp and nsProfileAccess.cpp.

3.2.1 nsProfileAccess.cpp

This class handles the low level read and write operations with the profile registry. When the browser is started up, nsProfileAccess will create a ProfileStruct object and fill it with the information from the registry file. This information includes the name, location, the last modified date, creation time, email address, etc. When changes to the profile need to be updated or a new profile is created, the UpdateRegistry function

will be called to write those changes to the registry file. This class also provides a set of "setter" and "getter" functions that forms a interface with the upper-level nsProfile class.

3.2.2 nsProfile.cpp

The nsProfile object accepts calls from other components to manage profiles. It can create, rename, copy, rename, and delete profiles. It also handles profile migration from former versions of Mozilla or Netscape as well as internationalization, which is not a trivial issue. But nsProfile object usually does not deal with the registry directly. Rather it calls its nsProfileAccess object to accomplish the tasks.

IV. Personal Security Manager (PSM) and Network Security Services (NSS)

4.1 Overview

PSM provides end-to-end security between the client, i.e., the browser and Web servers. PSM supports SSL (Secure Socket Layer) v2, v3 and TLS (Transport Layer Security). It provides a large variety of cipher suites for key exchange, digital signatures, bulk encryption, and data integrity. Its sophisticated user interface allows users to manage their passwords, cookies, certificates for mutual authentication and customize their security settings. For instance, users can disable SSL v2, weak ciphers with small key size, or approve certificates only from certain certificate authorities. The user interface makes it very easy to understand as long as the user has some basic knowledge about Internet security. PSM also supports most of the PKI (Public Key Infrastructure) specifications.

4.2 Implementation

There are two XPCOM shared libraries: *pki* and *ssl*. The *ssl* package links to NSS 3.2, handles all the SSL sockets, provides event handlers and appropriate warnings, defines and implements IDL interfaces for access to NSS libraries. The *ssl* package also

Supports embedding systems to use the cryptographic components without the user interface. Besides functionalities, the *ssl* package also offers good performance which is fast enough for disk encryption. The goal is to achieve 1MB per second for both encryption and decryption.

The *pki* package implements the user interface using XUL and related XPCOM objects.

4.3 PSM by Example

The following code fragments are from nsNTLMAuthModule.cpp. This is a good example to examine how the cryptographic module is reused. nsNTLMAuthModule.cpp implements DES and MD5 using NSS API. Here are some if its functions:

```
// set odd parity bit (in least significant bit position) static PRUint8
des_setkeyparity(PRUint8 x)

// build 64-bit des key from 56-bit raw key
static void des_makekey(const PRUint8 *raw, PRUint8 *key)

// run des encryption algorithm (using NSS)
static void des_encrypt(const PRUint8 *key, const PRUint8 *src, PRUint8 *hash)

// MD5 support code
static void md5sum(const PRUint8 *input, PRUint32 inputLen, PRUint8 *result)
```

If we take a closer look at the des_encrypt function, we will notice that this function does not implement the actual encryption code. In stead, it calls functions in the NSS (Network Security Services) package.

```
// run des encryption algorithm (using NSS)
static void des_encrypt(const PRUint8 *key, const PRUint8 *src, PRUint8 *hash)
{
    ...
    keyItem.data = (PRUint8 *) key;
    keyItem.len = 8;
    symkey = PK11_ImportSymKey(slot, cipherMech, PK11_OriginUnwrap, CKA_ENCRYPT,
                                          &keyItem, nsnull);
    if (!symkey)
    {
        NS_ERROR("no symkey");
        goto done;
  }
   . ..
   rv = PK11_CipherOp(ctxt, hash, (int *) &n, 8, (PRUint8 *) src, 8);
   if (rv != SECSuccess)
```

```
{
        NS_ERROR("des failure");
        goto done;
}
rv = PK11_DigestFinal(ctxt, hash+8, &n, 0);
if (rv != SECSuccess)
{
        NS_ERROR("des failure");
        goto done;
}
```

4.4 NSS API

NSS provides an open-source implementation of security libraries that can be reused by embedding applications. It supports SSL v2 and v3, TLS v1, PKCS #1, #3, #5, #7, #8, #9, #10, #11, #12, S/MIME for encrypted MIME data, X.509 v3 certificates, OCSP (The Online Certificate Status Protocol), PKIX Certificate and CRL Profile, and a suite of advanced ciphers such as AES, RSA, DSA, Triple DES, DES, Diffie-Hellman, RC2, RC4, SHA-1, MD2, MD5. NSS also provides tools to manage keys and security modules, and to debug and diagnose code.

The following functions are defined in security/nss/lib/pk11wrap:

PK11_Authenticate,
PK11_ChangePW,
PK11_CheckUserPassword,
PK11_CipherOp,
PK11_CloneContext,
PK11_ConfigurePKCS11,
PK11_CreateContextBySymKey,
PK11_CreateDigestContext,
PK11_DestroyContext,
PK11_DestroyTokenObject,
PK11_DigestBegin,
PK11_DigestOp,
PK11_DigestFinal,
PK11_DoesMechanism,
PK11_Finalize,
PK11_FindCertByIssuerAndSN,
PK11_FindCertFromDERCert...

V. Implementation of the Profile Loading Feature

5.1 Description

The first major feature of the USB Key Profile Manager is to automatically detect and

load profiles from a removable disk such as a USB drive. As I mention in the introductory section, the current version of Mozilla does not recognize profiles unless they are registered.

My profile loading feature solves this problem and helps to achieve location independence without extra copying and pasting work from the user.

5.2 The Process

The USB profile manager is implemented as a XPCOM component for Mozilla. When the browser is started up, the component will register itself with Mozilla and start running. It first checks how many usb storage media are mounted to the system, then loops through each mounted disk searching for profiles. The identification of a profile for now is a simplified process, i.e., all directories of which the name ends with ".slt" are considered valid Mozilla profiles.

Once a profile is found, it is added directly to Mozilla's profile registry.
If no USB profile is found, Mozilla's profile manager will act as usual.
If only one USB profile is located, the component will tell Mozilla to suppress the profile manager dialog at startup and use the USB profile by default.
If multiple are located, the profile manager dialog will be shown with all the profiles available, both on the local drive or the USB drive.

VI. Advanced Encryption Standard (AES)

I am considering using AES to encrypt profile folders so that they will be tamper-proof. AES is an iterated block cipher with a block size of 128 bits. It is considered a stronger encryption algorithm than DES (Data Encryption Standard). AES uses variable-length key size. Typical key sizes are 128-bit, 192-bit, and 256-bit. Depending upon the key length, AES runs through 10 to 14 rounds of encryption, each of which contains four operations: a byte-for-byte subsititution with an S-box, rearrangement of bytes using ShiftRow, MixColumn, and MixAddRoundKey.

VII. Conclusion and Future Tasks

Mozilla's profile manager allows user to customize their own settings of the browser. It provides great user interface and centralized file location so that users can easily access, examine, modify the content. However, if a regular user can do so, it is not too hard for an unauthorized user to do so either, especially when the profiles are stored on a removable disk, because Mozilla itself does not provide additional encryption and user authentication to protect those profiles.

Also, the current profile manager doesn't achieve location independence yet. This means that basically profiles created on one computer can only be used with that particular computer. If you want another computer to share the profile, you have to copy and paste files into the new profile folder.

The purpose of this project is to implement an add-on XPCOM component that can automatically detect, load, and unload profiles stored on a USB drive so that no footprint will be left on the local drive. Also, disk encryption and user authentication will be used to prevent unauthorized users from peak into and modify the profiles. Ideally, integrity check can also be done according to some user-specified conditions.

Mozilla is a very complex and yet well structured software system. In order for a new component to work smoothly with the entire system, many coding rules have to be followed and the component needs to use appropriate interfaces and offer a well-defined interface to export its own functionalities. One of the most difficult parts of this project is trying to change the behavior of the existing profile manager without changing the existing code. My current solutions is to emulate the behavior of the existing profile manager and communicate directly with the profile registry before the profile manager is initialized. I do keep a copy of the previous registry settings so that before the browser is shut down, the registry can be reset to what it was before the changes the USB key profile manager made.

During the course of CS 297, I got familiar with the overall structure of the Mozilla

browser with the focus on the implementation of Mozilla's profile manager and PSM. I also learned how to create XPCOM components, how to invoke the functions and services of other XPCOM components and how to create user interfaces using XUL (XML-Based User Interface Language). Through a large amount of research, I implemented the USB profile loading feature, the first major component of the USB Profile Manager.

The following is a list of the future tasks to accomplish:

1. If no profile exists either on the local or USB drive, provide a dialog for the user to create the very first profile, instead of creating one on the local drive by default.
2. Remove loaded USB profiles when the system is shut down.
3. Reset the profile settings to what they were before the changes made by the USB profile manager.
4. Lock the profile in use.
5. Disk encryption and user authentication. This can make the USB drive as a smartcard.
6. Handle profile migration and internationalization.
7. User interface for the user to choice whether integrity check on certain files is preferred.

Appendix A: The Code for the Performance Test of a USB Drive

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>
#include <cstdlib>
#include <fstream>
#include <ctime>
#include <string>
#include <sys/time.h>
using namespace std;

int main(int argc, char *argv[])
{
  ofstream outToUSB("/mnt/usbstick/out.txt");
  //ofstream outToUSB("out.txt");
```

```
 // read the first 1024 chars and repeatedly write it to USB stick.
 ifstream inFromPWD("main.cpp");

 // read from the file "out.txt" from the USB drive
 // file size = 97.7MB
 //ifstream inFromPWD("/mnt/usbstick/out.txt");
 //ifstream inFromPWD("out.txt");
 if(!inFromPWD)
 {
  cerr << "File open fails\n";
  exit(1);
 }

 const int buf_size = 32;
 char buffer[buf_size];
 int cnt = 5000000;

 inFromPWD.read(buffer, buf_size);

 // start timer
 timeval timer1;          // check /usr/include/time.h
 gettimeofday(&timer1, 0);
 cout << "Start at time (s): " << timer1.tv_sec << endl;

 //while(inFromPWD.read(buffer, buf_size))
 //{
   //outToUSB.write(buffer, inFromPWD.gcount());
   //cout.write(buffer, inFromPWD.gcount());
 //}
   //outToUSB.write(buffer, inFromPWD.gcount());
   //cout.write(buffer, inFromPWD.gcount());

 int bytes;
 for(int i = 0 ; i < cnt; i++)
 {
  outToUSB.write(buffer, buf_size);
  if(bytes > 0)
    cout << i << endl;
 }

 inFromPWD.close();
 outToUSB.close();

 // calculate time
 timeval timer2;
 gettimeofday(&timer2, 0);
 cout << "Stop at time (s): " << timer2.tv_sec << endl;
 cout << "\nElapsed time: " << (timer2.tv_sec - timer1.tv_sec)<< " s.\n";
 return EXIT_SUCCESS;
}
```

Appendix B: The Code for the Profile Loading Feature

```
#include "nsIGenericFactory.h"
#include "nsCOMPtr.h"
```

```
#include "nsXPCOM.h"
#include "nsIServiceManager.h"
#include "nsICategoryManager.h"
#include "nsIComponentManager.h"
#include "nsIObserver.h"
#include "nsMemory.h"
#include "nsIUSBProfileManager.h"
#include "nsXPCOMCID.h"
#include "nsISupportsPrimitives.h"
#include "nsIIOService.h"
#include "nsString.h"
#include "nsIRegistry.h"
#include "nsXPIDLString.h"
#include "nsIFile.h"
#include "nsILocalFile.h"
#include "nsAppDirectoryServiceDefs.h"
#include "nsDirectoryServiceUtils.h"
#include "nsDirectoryServiceDefs.h"
#include "nsIEnumerator.h"
#include "nsIWindowWatcher.h"
#include "nsIDialogParamBlock.h"
#include "nsIDOMWindowInternal.h"

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
# include <mntent.h>

static NS_DEFINE_CID(kSupportsCStringCID, NS_SUPPORTS_CSTRING_CID);

#define USBPROFILE_CID \
{ 0x777f7150, 0x4a2b, 0x4301, \
{ 0xad, 0x10, 0x5e, 0xab, 0x25, 0xb3, 0x22, 0xaa}}

#define USBPROFILE_ContractID "@azhou/USBProfile"

// Registry Keys

#define kRegistryYesString (NS_LITERAL_STRING("yes"))
#define kRegistryNoString (NS_LITERAL_STRING("no"))

#define kRegistryProfileSubtreeString (NS_LITERAL_STRING("Profiles"))
#define kRegistryCurrentProfileString (NS_LITERAL_STRING("CurrentProfile"))
#define kRegistryNCServiceDenialString (NS_LITERAL_STRING("NCServiceDenial"))
#define kRegistryNCProfileNameString (NS_LITERAL_STRING("NCProfileName"))
#define kRegistryNCUserEmailString (NS_LITERAL_STRING("NCEmailAddress"))
#define kRegistryNCHavePREGInfoString (NS_LITERAL_STRING("NCHavePregInfo"))
#define kRegistryHavePREGInfoString (NS_LITERAL_STRING("HavePregInfo"))
#define kRegistryMigratedString (NS_LITERAL_STRING("migrated"))
#define kRegistryDirectoryString (NS_LITERAL_STRING("directory"))
#define kRegistryNeedMigrationString (NS_LITERAL_STRING("NeedMigration"))
#define kRegistryMozRegDataMovedString (NS_LITERAL_STRING("OldRegDataMoved"))
#define kRegistryCreationTimeString (NS_LITERAL_CSTRING("CreationTime"))
#define kRegistryLastModTimeString (NS_LITERAL_CSTRING("LastModTime"))
#define kRegistryMigratedFromString (NS_LITERAL_CSTRING("MigFromDir"))
#define kRegistryVersionString (NS_LITERAL_STRING("Version"))
#define kRegistryVersion_1_0 (NS_LITERAL_STRING("1.0"))
```

```
#define kRegistryCurrentVersion (NS_LITERAL_STRING("1.0"))
#define kRegistryStartWithLastString (NS_LITERAL_CSTRING("AutoStartWithLast"))

#define PROFILE_MANAGER_URL
"chrome://communicator/content/profile/profileSelection.xul?manage=true"

const char* kDefaultOpenWindowParams = "centerscreen,chrome,modal,titlebar";

/* A mount table entry. */
struct mount_entry
{
  char *me_devname;                     /* Device node pathname, including "/dev/". */
  char *me_mountdir;                    /* Mount point directory pathname. */
  char *me_type;                        /* "nfs", "4.2", etc. */
  dev_t me_dev;                                         /* Device number of me_mountdir. */
  unsigned int me_dummy : 1;  /* Nonzero for dummy filesystems. */
  unsigned int me_remote : 1;   /* Nonzero for remote fileystems. */
  struct mount_entry *me_next;
};

#ifndef ME_DUMMY
# define ME_DUMMY(fs_name, fs_type) \
    (!strcmp (fs_type, "auto") \
     || !strcmp (fs_type, "autofs") \
     /* for Irix 6.5 */ \
     || !strcmp (fs_type, "ignore"))
#endif

#ifndef ME_REMOTE
# define ME_REMOTE(fs_name, fs_type) (strchr (fs_name, ':') != 0)
#endif

////////////////////// USBProfileManager ////////////////////////////

class nsUSBProfileManager: public nsIObserver,
                           public nsIUSBProfileManager
{
private:
   PRBool mLocked;

public:
nsUSBProfileManager();
virtual ~nsUSBProfileManager();

NS_DECL_ISUPPORTS

// The Observe function serves as the main function. That's when the component is loaded
// to Mozilla and started.
NS_DECL_NSIOBSERVER

NS_DECL_NSIUSBPROFILEMANAGER

private:
// helper method
// @param baseDir – the USB drive to search. For instance: /mnt/usbstick
NS_IMETHODIMP SearchForProfile(nsCOMPtr<nsIFile> baseDir);
```

```
/** @param dirEntry – A directory.
   @return PR_TRUE if is a profile, PR_FALSE otherwise.
   More sophiscated validation methods can be implemented later
*/
PRBool IsProfile(nsCOMPtr<nsIFile> dirEntry);

/** @param profileName – The name of the profile, which is actually the name of the

                parent folder of the profile folder xxx.slt
   @param profilePath – the full path name of the profile folder
*/
NS_IMETHODIMP LoadProfile(const PRUnichar *profileName, const PRUnichar *profilePath);

// record number of profiles
int numOfUSBProfiles;

// Mozilla's special registry file
nsCOMPtr<nsIFile> mNewRegFile;

// The profile to be used, needed when only one USB profile is located
nsString mUSBProfileName;

// the following fields are for resetting to the last state
nsString prevProfileName;

// record the previous startWithLastUsedProfile flag
PRBool prevStartMode;
};

nsUSBProfileManager::nsUSBProfileManager()
{
mLocked = PR_TRUE;
numOfUSBProfiles = 0;
NS_GetSpecialDirectory(NS_APP_APPLICATION_REGISTRY_FILE, getter_AddRefs(mNewRegFile));
mUSBProfileName = NS_LITERAL_STRING("");
NS_INIT_ISUPPORTS();
}

nsUSBProfileManager::~nsUSBProfileManager()
{
}

// TODO: need better way to verify. For ex. check whether some required files, dirs exist
PRBool nsUSBProfileManager::IsProfile(nsCOMPtr<nsIFile> dirEntry)
{
        nsCAutoString newPathName, ext;
        nsresult rv = dirEntry->GetNativeLeafName(newPathName);
        if(NS_SUCCEEDED(rv))
        {
                newPathName.Right(ext, 4);
                if(ext.Equals(NS_LITERAL_CSTRING(".slt")))
                        return PR_TRUE;
                else
                        return PR_FALSE;
        }
        return PR_FALSE;
}
```

```
NS_IMETHODIMP nsUSBProfileManager::SearchForProfile(nsCOMPtr<nsIFile> baseDir)
{
		PRBool hasMore = PR_FALSE;
		PRBool isDir;
		nsCOMPtr<nsISimpleEnumerator> dirIterator;
		nsresult rv = baseDir->GetDirectoryEntries(getter_AddRefs(dirIterator));
		if (NS_FAILED(rv)) return rv;

		rv = dirIterator->HasMoreElements(&hasMore);
		if (NS_FAILED(rv)) return rv;

		nsCOMPtr<nsIFile> dirEntry, parentDir;
		nsAutoString profileName, profilePath;

		while (hasMore)
		{
				rv = dirIterator->GetNext((nsISupports**)getter_AddRefs(dirEntry));
				if (NS_SUCCEEDED(rv))
				{
				   rv = dirEntry->IsDirectory(&isDir);
				   if (NS_SUCCEEDED(rv))
				   {
				      if (isDir)
				      {
				         if(IsProfile(dirEntry))
				         {
				                rv = dirEntry->GetParent(getter_AddRefs(parentDir));
				                if (NS_SUCCEEDED(rv))
				                {
				                    numOfUSBProfiles++;
				                   rv = parentDir->GetLeafName(profileName);
				                    rv = dirEntry->GetPath(profilePath);
				                // load the profile to the registry
				                rv = LoadProfile(profileName.get(), profilePath.get());

				                // record the profileName for auto launching
				                 if(NS_SUCCEEDED(rv) &&
				                mUSBProfileName.Equals(NS_LITERAL_STRING(""
				                                mUSBProfileName = profileName;
				                }
				         }
				         else
				                SearchForProfile(dirEntry);
				      }
				   }
				}
		    rv = dirIterator->HasMoreElements(&hasMore);
		    if (NS_FAILED(rv)) return rv;
		}
		 return rv;
}

// Set the directory value and add the entry to the registry tree.
NS_IMETHODIMP nsUSBProfileManager::LoadProfile(const PRUnichar *profileName, const PRUnichar
*profilePath)
{
```

```
    nsresult rv;
   // get the profile registry
nsCOMPtr<nsIRegistry> registry(do_CreateInstance(NS_REGISTRY_CONTRACTID, &rv));
if (NS_FAILED(rv)) return rv;

// TODO: need to close the registry???
rv = registry->Open(mNewRegFile);
if (NS_FAILED(rv)) return rv;

nsRegistryKey profilesTreeKey;

// Get the major subtree
rv = registry->GetKey(nsIRegistry::Common, kRegistryProfileSubtreeString.get(),
                                &profilesTreeKey);
if (NS_FAILED(rv))
{
        rv = registry->AddKey(nsIRegistry::Common,
        kRegistryProfileSubtreeString.get(), &profilesTreeKey);
        if (NS_FAILED(rv)) return rv;
}

 // add the new node to the registry
   nsRegistryKey profKey;

   rv = registry->AddKey(profilesTreeKey, profileName, &profKey);
      if (NS_FAILED(rv)) return rv;

      rv = registry->SetString(profKey, kRegistryMigratedString.get(), kRegistryYesString.get());

      if (NS_FAILED(rv)) return rv;

      registry->SetString(profKey, kRegistryNCProfileNameString.get(), profileName);
/*
      registry->SetString(profKey,
                  kRegistryNCServiceDenialString.get(),
                  profileItem->NCDeniedService.get());

      registry->SetString(profKey,
                  kRegistryNCUserEmailString.get(),
                  profileItem->NCEmailAddress.get());

      registry->SetString(profKey,
                  kRegistryNCHavePREGInfoString.get(),
                  profileItem->NCHavePregInfo.get());

      registry->SetLongLong(profKey,
                    kRegistryCreationTimeString.get(),
                    &profileItem->creationTime);

      registry->SetLongLong(profKey,
                    kRegistryLastModTimeString.get(),
                    &profileItem->lastModTime);
*/
     // Externalize the location
    rv = registry->SetString(profKey, kRegistryDirectoryString.get(), profilePath);

      if (NS_FAILED(rv)) {
```

```
            NS_ASSERTION(PR_FALSE, "Could not update profile location");
            return rv;
          }

        // TODO:     profileItem->ExternalizeMigratedFromLocation(registry, profKey);
        return NS_OK;
}

NS_IMPL_ISUPPORTS2(nsUSBProfileManager, nsIObserver, nsIUSBProfileManager);
NS_GENERIC_FACTORY_CONSTRUCTOR(nsUSBProfileManager)

////////////////// inplement nsIObserver /////////////////

NS_IMETHODIMP
nsUSBProfileManager::Observe(nsISupports *aSubject, const char *aTopic, const PRUnichar *aData)
{
nsresult rv;
nsCOMPtr<nsIComponentManager> aCompMgr;
NS_GetComponentManager(getter_AddRefs(aCompMgr));
if (!aCompMgr)
   return NS_ERROR_UNEXPECTED;

if(strcmp("http-startup", aTopic) == 0){
printf("*************************************************************\n");
printf("Observe http-startup: Registered...\n");
printf("Topic: %s\n", aTopic);
}

else if(strcmp("app-startup", aTopic) == 0){
printf("*************************************************************\n");
printf("Observe app-startup: Registered...\n");
printf("Topic: %s\n", aTopic);
}

else if(strcmp("xpcom-startup", aTopic) == 0){
printf("*************************************************************\n");
printf("*************************************************************\n");
printf("Observe xpcom-startup: Registered...\n");
printf("Topic: %s\n", aTopic);

                // register for "http-startup" event
   nsCOMPtr<nsIServiceManager> servman = do_QueryInterface((nsISupports*)aCompMgr, &rv);
   if (NS_FAILED(rv)) return rv;

   nsCOMPtr<nsICategoryManager> catman;
   servman->GetServiceByContractID(NS_CATEGORYMANAGER_CONTRACTID,
                           NS_GET_IID(nsICategoryManager),
                           getter_AddRefs(catman));
   if (NS_FAILED(rv)) return rv;
                                char* previous = nsnull;
               rv = catman->AddCategoryEntry("app-startup",
                       "USBProfileManager",
                       USBPROFILE_ContractID,
                       PR_TRUE,
                       PR_TRUE,
                       &previous);
```

```
   rv = catman->AddCategoryEntry("http-startup-category",
                            "USBProfileManager",
                            USBPROFILE_ContractID,
                            PR_TRUE,
                            PR_TRUE,
                            &previous);
   if (previous)
      nsMemory::Free(previous);

/* Linked list of mounted filesystems. */
struct mount_entry *me;

/* Read filesystem list. */
  struct mount_entry *mount_list;
  struct mount_entry **mtail = &mount_list;
    struct mntent *mnt;
    char *table = MOUNTED;
    FILE *fp;
    char *devopt;

    fp = setmntent (table, "r");
    if (fp == NULL)
      return NS_OK;                                                                    //NS_FAIL?

    while ((mnt = getmntent (fp)))
      {
                    me = (struct mount_entry *) malloc (sizeof (struct mount_entry));
                    me->me_devname = strdup (mnt->mnt_fsname);
                    me->me_mountdir = strdup (mnt->mnt_dir);
                    me->me_type = strdup (mnt->mnt_type);
                    me->me_dummy = ME_DUMMY (me->me_devname, me->me_type);
                    me->me_remote = ME_REMOTE (me->me_devname, me->me_type);
                    devopt = strstr (mnt->mnt_opts, "dev=");
                    if (devopt)
                      {
                        if (devopt[4] == '0' && (devopt[5] == 'x' || devopt[5] == 'X'))
                          me->me_dev = atoi (devopt + 6);
                        else
                          me->me_dev = atoi (devopt + 4);
                      }
                    else
                      me->me_dev = (dev_t) -1;      /* Magic; means not known yet. */

                    /* Add to the linked list. */
                    *mtail = me;
                    mtail = &me->me_next;
      }
                    me = mount_list;

    if (endmntent (fp) == 0)                                                    // What does this mean?
    {
    *mtail = NULL;

    while (mount_list)
      {
                    me = mount_list->me_next;
                    free (mount_list->me_devname);
```

```
                free (mount_list->me_mountdir);
                /* FIXME: me_type is not always malloced.  */
                free (mount_list);
                mount_list = me;
    }
  me = NULL;
 }
 *mtail = NULL;

                /* check if any /dev/sda devices, i.e. USB drives are mounted. */
                char* usb_dev = "/dev/sda";
                int len = strlen(usb_dev);
                for (; me; me = me->me_next)
                {
                                if(strncmp(me->me_devname, usb_dev, len) == 0)
                                {
                                                printf("%s %s\n", me->me_devname, me->me_mountdir);

                                                // For now, only consider one profile situation
                                                // look for the .slt folder which is the profile directory
                                                nsCOMPtr<nsIFile> baseDir;
                                                nsAutoString wideString;

                                                // TODO: change this!!!
                                                wideString.AssignWithConversion(me->me_mountdir);

                                                rv = NS_NewLocalFile(nsDependentString(wideString),
PR_TRUE, (nsILocalFile **)((nsIFile **)getter_AddRefs(baseDir)));
                                                if (NS_FAILED(rv)) return rv;

                                                // search for profiles and load
                                                SearchForProfile(baseDir);
                                }
                }
                printf("Usb Profiles: %d\n", numOfUSBProfiles);

                // get the profile registry
                nsCOMPtr<nsIRegistry> registry(do_CreateInstance(NS_REGISTRY_CONTRACTID,
&rv));
                if (NS_FAILED(rv)) return rv;

                rv = registry->Open(mNewRegFile);
                if (NS_FAILED(rv)) return rv;

                nsRegistryKey profilesTreeKey;

                // Get the major subtree
                rv = registry->GetKey(nsIRegistry::Common,
                  kRegistryProfileSubtreeString.get(), &profilesTreeKey);
                if (NS_FAILED(rv))
                {
                                rv = registry->AddKey(nsIRegistry::Common,
                                kRegistryProfileSubtreeString.get(), &profilesTreeKey);
                                if (NS_FAILED(rv)) return rv;
                }

                // Check if any profile exists
```

```
                nsCOMPtr<nsIEnumerator> enumKeys;
                rv = registry->EnumerateSubtrees( profilesTreeKey, getter_AddRefs(enumKeys));
                if (NS_FAILED(rv)) return rv;

                rv = enumKeys->First();
                if (NS_FAILED(rv)) return rv;

                if(NS_OK == enumKeys->IsDone())              // no items
                    {
                                // Open the profile manager dialog
                                printf("No profile!!!\n");
                                return rv;
                }

                if(numOfUSBProfiles == 1)
                {
                    //Get the current profile
                    nsXPIDLString tmpCurrentProfile;
                    rv = registry->GetString(profilesTreeKey,
                     kRegistryCurrentProfileString.get(),
                     getter_Copies(tmpCurrentProfile));

                    if (tmpCurrentProfile)
                      prevProfileName = NS_STATIC_CAST(const PRUnichar*, tmpCurrentProfile);

                      // Set the current profile
                      rv = registry->SetString(profilesTreeKey,
                      kRegistryCurrentProfileString.get(),
                      mUSBProfileName.get());
                      if (NS_FAILED(rv)) return rv;

                     // Get the StartWithLastProfile flag
                    PRInt32 tempLong;
                      rv = registry->GetInt(profilesTreeKey,
                     kRegistryStartWithLastString.get(), &tempLong);
                     if (NS_SUCCEEDED(rv))
                                prevStartMode = tempLong;

                    // Set the StartWithLastProfile flag to true only when
                    rv = registry->SetInt(profilesTreeKey,
                  kRegistryStartWithLastString.get(), PR_TRUE);
                   if (NS_FAILED(rv)) return rv;
                }

                // TODO: lock
}

// NEED TO FREE mount_entry *me? //
return NS_OK;
}

static NS_METHOD nsUSBProfileManagerRegistration(
  nsIComponentManager *aCompMgr,
  nsIFile *aPath,
  const char *registryLocation,
  const char *componentType,
  const nsModuleComponentInfo *info)
```

```
{
   nsresult rv;
   nsCOMPtr<nsIServiceManager> servman = do_QueryInterface((nsISupports*)aCompMgr, &rv);
   if (NS_FAILED(rv))
      return rv;

   nsCOMPtr<nsICategoryManager> catman;
   servman->GetServiceByContractID(NS_CATEGORYMANAGER_CONTRACTID,
                           NS_GET_IID(nsICategoryManager),
                           getter_AddRefs(catman));
   if (NS_FAILED(rv))
      return rv;

   char* previous = nsnull;
   rv = catman->AddCategoryEntry("profile-do-change",

                           "USBProfileManager",

                                                                        USBPROFILE_ContractID,
                           PR_TRUE,
                           PR_TRUE,
                           &previous);
   if (previous)
      nsMemory::Free(previous);

   return rv;
}


static NS_METHOD nsUSBProfileManagerUnregistration(nsIComponentManager *aCompMgr,
nsIFile *aPath,
const char *registryLocation,
const nsModuleComponentInfo *info)
{
nsresult rv;

nsCOMPtr<nsIServiceManager> servman = do_QueryInterface((nsISupports*)aCompMgr, &rv);
if (NS_FAILED(rv))
return rv;


nsCOMPtr<nsICategoryManager> catman;
servman->GetServiceByContractID(NS_CATEGORYMANAGER_CONTRACTID,
NS_GET_IID(nsICategoryManager),
getter_AddRefs(catman));

if (NS_FAILED(rv))
return rv;

rv = catman->DeleteCategoryEntry("xpcom-startup",
"USBProfileManager",
PR_TRUE);

rv = catman->DeleteCategoryEntry("content-policy",
"USBProfileManager",
PR_TRUE);
```

```
return rv;
}

////////////////// inplement nsIUSBProfileManager /////////////////
// Not implemented

////////////////// nsModuleComponentInfo //////////////////////

static const nsModuleComponentInfo components[] =
{
{ "USBProfile",
USBPROFILE_CID,
USBPROFILE_ContractID,
nsUSBProfileManagerConstructor,
nsUSBProfileManagerRegistration,
nsUSBProfileManagerUnregistration
}
};

NS_IMPL_NSGETMODULE(USBProfileModule, components)
```

References

[2003] "Creating XPCOM Components", Retrieved on 1/12/04, from http://www.mozilla.org/projects/xpcom/book/cxc/.
[2003] Rapid Application Development with Mozilla. Nigel Mcfarlane. Prentice Hall. 2003.
[2003] Running Linux. Matthias Kalle Dalheimer, Terry Dawson, Lar Kaufman, Matt Welsh. O'Reilly. 2003.
[2003] "Netscape Portable Runtime", Retrieved on 1/8/04, from http://www.mozilla.org/projects/nspr/.
[2003] "XPCOM". Retrieved on 1/8/04, from http://www.mozilla.org/projects/xpcom/.
[2003] “Linux and USB 2.0”. By David Brownell. Retrieved from http://www.linux-usb.org/usb2.html#current, on 3/1/04.
[2003] "Security Projects", Retrieved on 1/8/04, from http://www.mozilla.org/projects/security/.
[2003] Understanding the Linux Kernel. Daniel P. Bovet, Marco Cesati. O'Reilly. 2003.
[2003] Linux Device Drivers. Jonathan Corbet, Alessandro Rubini. O'Reilly. 2003.
[2002] The Cathedral & the Bazaar. Eric S. Raymond. O'Reilly. 2002.
[1999] "State-of-the-art ciphers for commercial applications". Computers & Security. Preneel, Bart. 1999.
Red Hat Linux 7.3: The Official Red Hat Linux Reference Guide. Retrieved on 1/8/04, from http://linux.web.cern.ch/linux/redhat73/documentation/redhatcd/RH-DOCS/rhl-rg-en-7.3/ch-ext3.html.